Recent Advances on the Algorithmic Optimization of Robot Motion

James E. Bobrow, Frank C. Park, and Athanasios Sideris

Department of Mechanical and Aerospace Engineering
University of California, Irvine
Irvine, CA 92697
{jebobrow, asideris}@uci.edu

Summary. An important technique for computing motions for robot systems is to conduct a numerical search for a trajectory that minimizes a physical criteria like energy, control effort, jerk, or time. In this paper, we provide example solutions of these types of optimal control problems, and develop a framework to solve these problems reliably. Our approach uses an efficient solver for both inverse and forward dynamics along with the sensitivity of these quantities used to compute gradients, and a reliable optimal control solver. We give an overview of our algorithms for these elements in this paper. The optimal control solver has been the primary focus of our recent work. This algorithm creates optimal motions in a numerically stable and efficient manner. Similar to sequential quadratic programming for solving finite-dimensional optimization problems, our approach solves the infinite-dimensional problem using a sequence of linear-quadratic optimal control subproblems. Each subproblem is solved efficiently and reliably using the Riccati differential equation.

1 Introduction

For many biological systems, it has long been observed that motion generation can likely be the result of a minimization process. The objective function used has been characterized by a physical criteria like energy, control effort, jerk, or time. Unfortunately, to date the algorithms that generate such optimal motions have been successfully used on only the simplest of robots. The need for such an algorithm is increasing dramatically since many new walking, crawling, hopping machines, rehabilitation devices, and free-flying air and space systems are currently under development. All of these devices will benefit from a numerically stable and efficient algorithm that produces optimal movements for them.

We are interested in obtaining solutions to optimal control problems for systems of the form

$$\dot{x} = f(x(t), u(t)), \tag{1}$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n \in C^1$ (continuously differentiable) and $x(0) = x_o$. We assume that the optimal control cost functional has the form

Fig. 1. The left-hand system was used for Case 1, a fully actuated robot. Case 2 is the same system, but with the base joint unactuated. The center system represents Case 3, which is an application to human step rehabilitation. The right system is a hopping machine for Case 4.

$$\underset{u(t)}{\text{Minimize}}\ J(u(t)) = \phi(x(t_f)) + \int_0^{t_f} L(x(t), u(t), t)\ dt, \tag{2}$$

subject to (1) with $\phi : \mathbb{R}^n \to \mathbb{R} \in C^1$ and $L : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R} \in C^1$. Although the Maximum Principle [3] provides the optimality conditions for the solution to (2), it is not suitable for numerical computation. Because of the importance in solving these problems, many numerical algorithms and commercial software packages have been developed to solve them since the 1960's [1]. Most of the existing algorithms do not have adequate numerical stability properties and are too slow computationally to solve optimal control problems for current multibody systems. As a means to discuss the numerical features of algorithms, we provide example solutions in four case studies. These examples demonstrate the strength and limitations of current numerical algorithms.

Figure 1 shows model systems used for four case studies in this paper. Case 1 is a minimum effort control of a fully actuated robot. We have found that with care in the choice of basis functions, direct methods can be tailored to adequately solve this problem. Case 2 is an underactuated robot. We have found that even with exact gradients of the dynamics, direct methods have numerical problems from round-off errors during the simulation of the motion. Case 3 is an application to human leg step rehabilitation [21]. We experienced even more numerical problems for this problem due to the added ground constraint. Finally, Case 4 is a simplified gas actuated hopping machine. We found it difficult to achieve stable convergence with existing methods for this case. The cause was numerical integration errors introduced at the sudden change in the dynamics between the stance phase and the flight phase, and the fact that the times for the switch from stance to flight were not known apriori. The approach developed in this paper (see also [14]) efficiently solves Case 4 , and we feel that it has great potential for application to general optimal control problems for robot systems.

1.1 First order necessary conditions for the solution of the optimal control problem

In order to discuss the solution to Cases 1-4, we first briefly summarize the first order necessary conditions for the optimal control of a general nonlinear system (see [3] for more details). First define the Hamiltonian as

$$H(x, u, \lambda, t) \equiv L(x, u, t) + \lambda^T f(x, u), \tag{3}$$

where L and f were defined in (1) and (2). Then in (3), $\lambda(t)$ is chosen to satisfy the costate or adjoint equations

$$\dot{\lambda} = -H_x(x_o(t), u_o(t), \lambda(t), t), \tag{4}$$

where H_x and H_u (used below) denote partial derivatives of H with respect to x and u respectively, and the boundary conditions are

$$\lambda(t_f) = \phi_x^T(x_o(t_f)).$$

Let $u_o(t)$ be a nominal control, $x_o(t)$ and $\lambda_o(t)$ be the corresponding solutions to (1) and (4), respectively. For general problems, the first order necessary conditions for a local minimum of J require that $H(x_o(t), u_o(t), \lambda(t), t)$ be minimized with respect to $u_o(t)$ subject to any constraints on it. For unconstrained controls u, the condition on H is

$$H_u(x_o(t), u_o(t), \lambda(t), t) = 0.$$

Note that both H_x and H_u require differentiation of the state equations (1) with respect to x and u and evaluation of these derivatives along the solution $(x_o(t), u_o(t))$. For multibody dynamic systems with more than a few degrees of freedom, the derivatives are generally not available due to the complexity of the equations of motion. However, in [15], the sensitivity algorithms based on matrix exponentials are developed specifically for this purpose. A brief introduction to that work is presented next.

1.2 Geometric Tools for Multibody Systems Analysis

To represent robot systems and their dynamics, we use a set of analytical tools for multibody systems analysis based on the mathematics of Lie groups and Lie algebras [13, 11]. In the traditional formulation, a rigid motion can be represented with the Denavit-Hartenberg parameters as a 4x4 homogeneous transformation $T(\theta, d) \in$ SE(3), where θ is the rotation about the z-axis and d is the translation along it. For a prismatic joint, d varies while θ is held constant. For a revolute joint, θ varies while d is held constant. With the geometric formulation, for either type of joint the transformation has the form

$$T(\theta, d) = e^{Ax} M,$$

where $x = \theta$ for a revolute joint or $x = d$ for a prismatic joint, A contains the joint axis or direction, and M is a constant ($M = T(0, d)$ for a revolute joint, $M = T(\theta, 0)$ for a prismatic joint.) This exponential mapping and its inverse have explicit formulas: $\exp : se(3) \to SE(3)$ and its inverse $\log : SE(3) \to se(3)$ [13]; here se(3) denotes the Lie algebra of SE(3). Although SE(3) is not a vector space,

- **Initialization**

$$V_0 = \dot{V}_0 = \mathcal{W}_{n+1} = 0$$

- **Forward recursion: for** $i = 1$ **to** n **do**

$$\begin{aligned} T_{i-1,i} &= M_i e^{S_i q_i} \\ V_i &= \mathrm{Ad}_{T_{i-1,i}^{-1}}(V_{i-1}) + S_i \dot{q}_i \\ \dot{V}_i &= S_i \ddot{q}_i + \mathrm{Ad}_{T_{i-1,i}^{-1}}(\dot{V}_{i-1}) + \left[\mathrm{Ad}_{T_{i-1,i}^{-1}}(V_{i-1}), S_i \dot{q}_i\right] \end{aligned}$$

- **Backward recursion: for** $i = n$ **to** 1 **do**

$$\begin{aligned} \mathcal{W}_i &= \mathrm{Ad}^*_{T_{i,i+1}^{-1}}(\mathcal{W}_{i+1}) + J_i \dot{V}_i - \mathrm{ad}^*_{V_i}(J_i V_i) \\ \tau_i &= S_i^T \mathcal{W}_i \end{aligned}$$

Fig. 2. The POE recursive Newton-Euler inverse dynamics algorithm.

se(3) is: the log formula provides a set of *canonical coordinates* for representing neighborhoods of SE(3) as open sets in a vector space.

The derivative of the exponential map with respect to the joint displacement x is just $\frac{dT}{dx} = Ae^{Ax}M$. In the coding of multibody dynamics algorithms, the exponential is the lowest level primitive required for all computations. One never needs to deal with sine and cosine terms or with making a distinction for each joint type.

The use of matrix exponentials to represent the link to link transformations for robot systems allows one to clarify the kinematic and dynamic equations. In the case of open chains containing prismatic or revolute joints, the forward kinematics can be written as a product of matrix exponentials [2]. Specifically, given a choice of inertial and tool reference frames, and a zero position for the mechanism, the forward kinematics can be written uniquely as

$$T_{0n}(q_1, \ldots, q_n) = e^{A_1 q_1} \cdots e^{A_n q_n}$$

where $q_1, \ldots, q_n$ are joint variables, and $A_1, \ldots, A_n \in \mathrm{se}(3)$. The kinematics of closed chains can be obtained by further adding a set of algebraic constraints.

In order to determine optimal motions for the multibody systems of interest, a complete dynamic model is needed. In [13] a Lie group formulation of the dynamics has been developed, in which closed-form expressions for the inertia matrix and Coriolis terms are available. Using this representation, the forward and inverse dynamics can also be computed efficiently with $O(n)$ recursive algorithms. The inverse dynamics algorithm is shown in Figure 2. In this algorithm, $V_i \in \mathrm{se}(3)$ is the linear and angular velocity of link i, $\mathcal{W}$ is the applied force and moment, J is a 6x6 matrix of mass and inertia, S_i is the joint screw, and Ad and ad are standard operators from differential geometry [11]. A useful computational feature of this algorithm is that no distinction needs to be made for revolute or prismatic joints. In [15], this algorithm was extended to forward and inverse dynamics of partially actuated systems, and to produce the derivatives needed for many optimal control solvers, as discussed in the previous section.

Given the ability to compute the dynamics and derivatives of relatively complex systems, we now discuss some representative examples. The following case studies

demonstrate both successes and difficulties that we have encountered in applying the optimality conditions of Section 1.1 to multibody systems problems.

2 Some representative case studies

2.1 Case 1: Fully Actuated Robot

Consider the case of finding the minimum effort control which moves the two link planar robot shown in Figure 1 from an outstretched horizontal position to a vertical position. Assume that both joints of the arm be actuated, and let the cost function be:

$$J = c_1||q(t_f) - q_d||^2 + c_2||\dot{q}(t_f)||^2 + \frac{1}{2}\int_0^{t_f} ||u||^2 dt, \tag{5}$$

where $q \in \mathbb{R}^2$ are the joint angles, $q_d = [\frac{\pi}{2}, 0]^T$, are the desired final joint positions in radians, $u \in \mathbb{R}^2$ are the corresponding joint torques, and $c_1 = c_2 = 100$ reflects the desire to reach the final vertical configuration with little error.

We used several approaches to solve this optimal control problem. The most straight forward approach, called the "shooting method," is to cleverly find the initial costate, $\lambda(0)$, such that when the state and costate equations are integrated forward from $t = 0$ to $t = t_f$ with $H_u = 0$, the proper final condition on the costate is satisfied ($\lambda(t_f) = \phi_x^T(x_o(t_f))$. Unfortunately, because the costate equations are not stable and highly sensitive to the initial conditions [3], the shooting method failed when applied to this problem. A second, more successful, approach used by several researchers [5, 4, 10, 12] in robotics is to approximate the motion of the joints using basis functions such as splines or a truncated Fourier Series. For instance, we used quintic B-spline polynomials with 12 uniformly spaced knot intervals to parameterize our solution as $q = q(t, P)$ with $P \in \mathbb{R}^2 \times \mathbb{R}^{12}$ being the amplitude of the spline basis functions. For any choice of P, we can compute the required control $u(t)$ by differentiating $q(t, P)$ with respect to time to obtain $\dot{q}$ and $\ddot{q}$ and, evaluating the equations of motion

$$M(q)\ddot{q} + h(\dot{q}, q) = u. \tag{6}$$

In order to use this "direct approach," we guessed an initial motion that kept the second link aligned with the first with $q_2(t) = 0$, and moved the first link smoothly from $q_1(0) = 0$ to $q_1(t_f) = \pi/2$, with $t_f = 2$ seconds. We then computed $J(u(P))$ in (5) and its gradient ∇J_P using adaptive Simpson quadrature. In this case, the integrand is known explicitly throughout the integral since all the terms in (6) are known explicitly in terms of P from the joint angles $q = q(t, P)$. Given $J(P)$ and its gradient, we could easily minimize it over P using Matlab's BFGS [9] algorithm in the function "fminunc."

Figure 3 shows the locally optimal solution found to this problem using the parameter optimization approach mentioned above. The frames are spaced at equal intervals in time, with $t_f = 2$ seconds. At first the robot allows gravity to take over and it swings down while folding up the second link. It then swings the first joint into the upward posture. A small pumping motion is applied to the second link in order to move it into the vertical posture. The initial value of the effort term in the cost function was 73.6 and the final value was 9.9. The computation time for this problem was about 2 minutes on a PIII-800 PC.

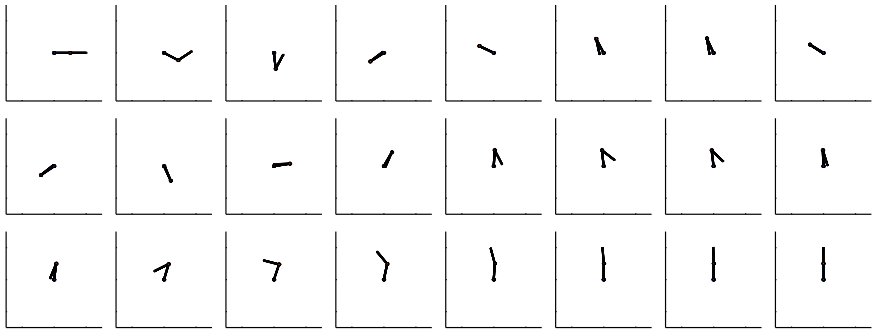


Fig. 3. Final path for fully-actuated planar 2R problem.

We have used this basic approach with our dynamics tools to solve a weight-lifting problem for a much more complex Puma 762 robot in [20], where we tripled the payload above the manufacturers specifications. Even though the solution to the above problem was fairly stable numerically, we still needed to choose an appropriate set of basis functions in order to approximate the numerical solution. In this paper, we develop an approach that does not use any basis functions to approximate the solution.

2.2 Case 2: Underactuated Robot

The optimal control problem becomes less numerically stable and more difficult to solve when the system is underactuated. For instance, suppose the motor attached to the first joint of the above 2R robot is disconnected. The system then only has a motor at its elbow and is often called the Acrobot, which has been studied by Spong [17] and others. Consider the swing-up motion problem where the system starts from a hanging downward posture and the optimal control problem is to find an open-loop control torque for the elbow, if one exists, that drives the system to the upward posture of the previous example. This case is much more challenging than the previous one because we can no longer use (6) to compute $u(P)$ because the system is underactuated.

Choose the same objective function as (5), except the control u is now a scalar. One way to approach the problem is approximate the control with a set of basis functions and integrate the 4 state equations in order to evaluate (5). Any gradient-based numerical optimization will need the both the value of $J(P)$ and its gradient $\nabla_p J$. Assuming that the state, costate, and boundary conditions are satisfied, the required derivative is

$$\frac{dJ}{dp_i} = \int_0^{t_f} H_u(x_o(t), u_o(t), \lambda(t), t) \frac{du}{dp_i}\, dt. \tag{7}$$

This derivative is valid for any $(x_o(t), u_o(t))$ even if they are not optimal [3]. Then, in order to evaluate the objective function and its gradient for use in any gradient-based nonlinear optimization algorithm, the following steps must be performed:

- Select a set of basis functions and parameters to define $u_o(t, P)$.
- Integrate the differential equations (1) of motion from 0 to t_f to obtain $x_o(t)$ and $J(P)$.
- Evaluate the the costate (4) boundary conditions $\lambda(t_f) = \phi_x^T(x_o(t_f))$ and integrate the costate equations backwards in time from t_f to 0 to obtain $\lambda(t)$.
- Evaluate the gradient of J using (7)

We used this method to solve the Acrobot swing-up problem with one modification. That is, instead of viewing the control u_2 as the torque at joint 2, we defined the motion of joint q_2 with our spline functions and considered that to be the control in (1). In doing so, the actual joint torque $u_2(t)$ can be computed algebraically from (6) assuming that $(q, \dot{q})$ are known. The advantage of doing this is that the state reduces to 2 dimensions in this case $x = [q_1, \dot{q}_1]^T$ since the motion of the second joint is known from the parameterization.

Note that it is not apparent what, if any, elbow motion will drive the system to the desired final configuration. Our initial guess for the elbow motion was very poor. We did not move the elbow joint at all during the motion, and let the system move like a rigid pendulum would with an initial condition $q_1(0) = -1.0, \dot{q}(0) = q_2(0) = \dot{q}_2(0) = 0.0$. Figure 4 shows the final motion obtained using Matlab's nonlinear parameter optimization with gradients computed as described above. The motion produced is similar to those proposed by Spong [17], in which the lower link pumps energy into the system and this energy causes the first link to move into the vertical position. In addition to this example, we have used this basic approach to solve for much more complex optimal high-dive motions for a human-like diver in [16].

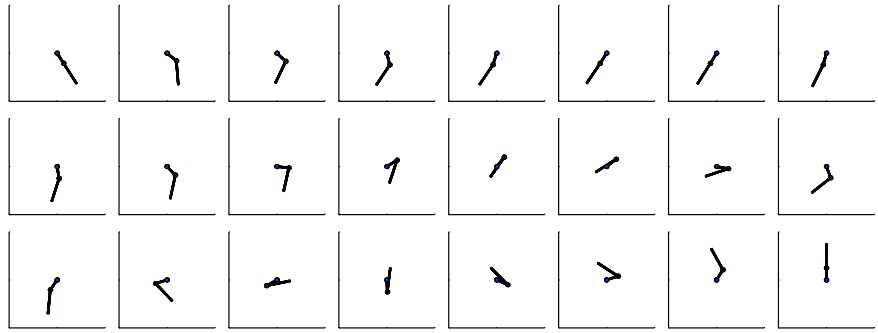


Fig. 4. Optimal swing up motion for an Acrobot with $q_1(0) = -1.0$.

When we computed the above solution to the underactuated Acrobot, we did not expect numerical difficulties, since we had the exact gradient of the objective function and the optimization algorithm has well-established convergence properties for this case [9]. However, we did encounter some numerical problems and had to adjust some of the tolerances in the optimizer in order to achieve convergence, and the computation time, even in the best of cases (about 5 minutes on a PIII-800 PC), was much longer than in the previous example. The problem was that the round-off errors encountered during the numerical solution of (1), (4), and (7) lead

to large relative errors in the gradient when the algorithm is near convergence. The algorithm developed in this paper alleviates these difficulties.

2.3 Case 3: Underactuated systems with contact constraints- Human step training example

One important application of our proposed algorithm is the generation of optimal inputs for the robotic rehabilitation of paralyzed individuals [22]. In [21] we examined a method to control the stepping motion of a paralyzed person suspended over a treadmill (see Fig. 1) using a robot attached to the pelvis. Leg swing motion was created by moving the pelvis without contact with the legs. The problem is formulated as an optimal control problem for an underactuated articulated chain. Similar to the underactuated Acrobot, the optimal control problem is converted into a discrete parameter optimization and a gradient-based algorithm is used to solve it.

To simulate a paralyzed person, a dynamic model for a branched kinematic chain was used approximate the kinematics and dynamics of a human subject. For the swing hip, knee and ankle joints, a torque was applied to simulate the stiffness of passive tissue, but no torque from the muscles since the person is assumed to be paralyzed. A total of 32 B-spline parameters were used in the optimization to specify the motion of the swing hip. This problem differed from the Acrobot because we had to constrain the motion of the foot to avoid contact with the ground, and the motion of the legs to avoid contact with each other. We used penalty functions to enforce these collision avoidance constraints.

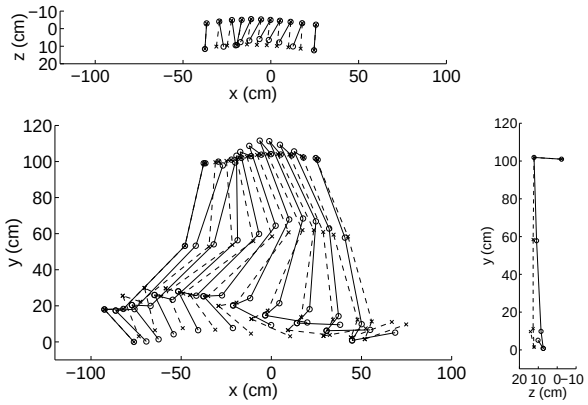


Fig. 5. Motion of the pelvis can be used to create motion for a paralyzed swing leg. The solid lines show gait which results from optimal motion of the pelvis, and the dashed lines are the gait recorded from the motion capture system.

In the optimization results shown in Figure 5, we found the motion of the swing hip that produced a step for the swing leg that was as close as possible to a normal

human gait. The optimal control found from our algorithm lifted the swing hip to avoid collision between the swing leg and the ground. At the same time, it twisted the pelvis to pump energy into the paralyzed leg and moved the leg close to the desired final configuration, while avoiding collision between the legs. Thus we found a strategy that could achieve repetitive stepping by shifting the pelvis alone. The optimized, pelvic motion strategies are comparable to "hip-hiking" gait strategies used by people with lower limb prostheses or hemiparesis.

Even though there were relatively few parameters (32) in the optimization, it was not numerically stable and took approximately 4 hours to converge. The problems were again due to round-off errors introduced by the computation of the gradient in (7). The penalty functions for obstacle avoidance exacerbated the problem since they effectively created a "stiff" system of differential equations. The above results only considered the swing phase of the gait cycle. In our initial attempts to combine the stance phase with the swing phase in the optimal control solution were numerically unstable and did not converge to a solution.

Based on our initial results from the simple hopping machine considered in Case 4, we believe that with our new algorithm it will be possible to combine the stance and swing phases and reliably compute an optimal motion for Case 3 in just a few minutes of computation time. This would make it possible to compute an optimal motion for each patient in a clinical setting.

2.4 Case 4: Minimum Fuel Hopping

In order to explore the difficulties associated with the change in dynamics between the stance phase and swing phase of motion mentioned in Case 3, we considered a simple one-dimensional hopping system shown in Figure 1. This system is driven by a pneumatic actuator, with the location of the piston relative to the mass under no external loading defined as y_p. After contact occurs with the ground with $y \leq y_p$, the upward force on the mass from actuator can be approximated by a linear spring with $F = k(y_p - y)$, where k is the spring constant. The position y_p can be viewed as the unstretched spring length and it can be easily changed by pumping air into or out of either side of the cylinder. The equations of motion for the mass are $m\ddot{y} = F(y, y_p) - mg$, where mg is the force due to gravity, and $F(y, y_p) = \begin{cases} 0 & y > y_p \\ k(y_p - y) & \text{otherwise.} \end{cases}$ Note that in this case $F(y, y_p)$ is not differentiable at $y = y_p$, and gradient-based methods will have difficulties with this. However, the discontinuity in the derivative can easily be smoothed. For instance, let the spring compression be $e = y_p - y$ and choose an $\alpha > 0$, then

$$F(e) = \begin{cases} 0 & 0 > e \\ \frac{k}{2\alpha} e^2 & 0 \leq e < \alpha \\ ke - \frac{k\alpha}{2} & \text{otherwise} \end{cases}$$

is C^1. The final equation of motion for this system relates the air flow into the cylinder, which is the control $u(t)$, to the equilibrium position y_p of the piston. Assume for the following that the equation $\dot{y}_p = u$ approximates this relationship.

When the hopping machine begins its operation, we are interested in starting from rest, and reaching a desired hop height y_N^o at time t_f. If we minimize

$$J(u) = \frac{1}{2} q_{fin} (y(N) - y_N^o)^2 + \dot{y}(N)^2 + \frac{t_f}{2N} \sum_{n=0}^{N-1} \left[q\ y_p(n)^2 + r\ u(n)^2 \right], \qquad (8)$$

the terms outside the summation reflect the desire to reach the height at time t_f with zero velocity, and the terms inside the summation reflect the desire to minimize the gas used to achieve this. The weighting on y_p is used to keep the piston motion within its bounds. We first attempted to solve this problem by parameterizing the control $u(t)$ with B-splines and using the basic steps used in Cases 2 and 3. Even after considerable tweaking of tolerances, the gradient-based algorithm would not converge. This drove us to develop the algorithm described in the next section.

3 Problem Formulation and Background Results

We assume that the dynamic system defined by (1) and the performance measure (2) have been descretized by a suitable numerical integration scheme. To simplify the notation, we use the same function and variable names for the discrete-time versions of the continuous-time variables. A more detailed discussion of this material can be found in [14].

$$\underset{u(n),\ x(n)}{\text{Minimize}}\ J = \phi(x(N)) + \sum_{n=0}^{N-1} L(x(n), u(n), n) \tag{9}$$

$$\text{subject to} \qquad x(n+1) = f(x(n), u(n)); \quad x(0) = x_0 \tag{10}$$

We further assume a quadratic performance index, namely:

$$L(x(n), u(n), n) = \frac{1}{2}[x(n) - x^o(n)]^T Q(n)[x(n) - x^o(n)] + [u(n) - u^o(n)]^T R(n)[u(n) - u^o(n)] \tag{11}$$

and

$$\phi(x) = \frac{1}{2}[x - x^o(N)]^T Q(N)[x - x^o(N)] \tag{12}$$

In (11) and (12), $u^o(n)$, $x^o(n)$, $n = 1, \ldots\ N$ are given control input and state offset sequences. In standard optimal regulator control problem formulations, $u^o(n)$, $x^o(n)$ are usually taken to be zero with the exception perhaps of $x^o(N)$, the desired final value for the state. The formulation considered here addresses the more general optimal tracking control problem and is required for the linear quadratic step in the proposed algorithm presented in Section 3.2.

3.1 First Order Optimality Conditions

We next briefly review the first order optimality conditions for the optimal control problem of (9) and (10), in a manner that brings out certain important interpretations of the adjoint dynamical equations encountered in a control theoretic approach and Lagrange Multipliers found in a pure optimization theory approach such as that mentioned in Section 1.1.

Let us consider the *cost-to-go*:

$$J(n) \equiv \sum_{n=k}^{N-1} L(x(k), u(k), k) + \phi(x(N)) \tag{13}$$

with L and ϕ as defined in (11) and (12) respectively. We remark that $J(n)$ is a function of $x(n)$, and $u(k)$, $k = n, \ldots, N-1$ and introduce the sensitivity of the cost to go with respect to the current state:

$$\lambda^T(n) = \frac{\partial J(n)}{\partial x(n)} \tag{14}$$

Since

$$J(n) = L(x(n), u(n), n) + J(n+1), \tag{15}$$

we have the recursion:

$$\begin{aligned} \lambda^T(n) &= L_x(x(n), u(n), n) + \lambda^T(n+1) f_x(x(n), u(n)) \\ &= [x(n) - x^o(n)]^T Q(n) + \lambda^T(n+1) f_x(x(n), u(n)) \end{aligned} \tag{16}$$

by using (11) and where L_x and f_x denote the partials of L and f respectively with respect to the state variables. The previous recursion can be solved backward in time ($n = N-1, \ldots, 0$) given the control and state trajectories and it can be started with the final value:

$$\lambda^T(N) = \frac{\partial L(N)}{\partial x(N)} = [x(N) - x^o(N)]^T Q(N) \tag{17}$$

derived from (12). We now compute the sensitivity of $J(n)$ with respect to the current control $u(n)$. Clearly from (15),

$$\begin{aligned} \frac{\partial J(n)}{\partial u(n)} &= L_u(x(n), u(n), n) + \lambda^T(n+1) f_u(x(n), u(n)) \\ &= [u(n) - u^o(n)]^T R(n) + \\ &\quad \lambda^T(n+1) f_u(x(n), u(n)) \end{aligned} \tag{18}$$

since $\frac{\partial J(n+1)}{\partial u(n)} = \frac{\partial J(n+1)}{\partial x(n+1)} \cdot \frac{\partial x(n+1)}{\partial u(n)} = \lambda^T(n+1) f_u(x(n), u(n))$. In (19) L_u and f_u denote the partials of L and f respectively with respect to the control variables and (11) is used.

Next note that $\frac{\partial J}{\partial u(n)} = \frac{\partial J(n)}{\partial u(n)}$ since the first n terms in J do not depend on $u(n)$. We have then obtained the gradient of the cost with respect to the control variables, namely:

$$\nabla_u J = \left[\frac{\partial J(0)}{\partial u(0)} \quad \frac{\partial J(1)}{\partial u(1)} \quad \cdots \quad \frac{\partial J(N-1)}{\partial u(N-1)} \right]. \tag{19}$$

Assuming u is unconstrained, the first order optimality conditions require that

$$\nabla_u J = 0. \tag{20}$$

We remark that by considering the Hamiltonian

$$H(x, u, \lambda, n) \equiv L(x, u, n) + \lambda^T f(x, u), \tag{21}$$

we have that $H_u(x(n), u(n), \lambda(n+1), n) \equiv \frac{\partial J}{\partial u(n)}$, i.e. we uncover the generally known but frequently overlooked fact that the partial of the Hamiltonian with respect to the control variables u is the gradient of the cost function with respect to u. We emphasize here that in our approach for solving the optimal control problem, we take the viewpoint of the control variables $u(n)$ being the independent variables of the problem since the dynamical equations express (recursively) the state variables in terms of the controls and thus can be eliminated from the cost function. Thus in taking the partials of J with respect to u, J is considered as a function $u(n)$, $n = 0, \ldots, N-1$ alone, assuming that $x(0)$ is given. With this perspective, the problem becomes one of unconstrained minimization, and having computed $\nabla_u J$, Steepest Descent, Quasi-Newton, and other first derivative methods can be brought to bear to solve it. However, due to the large-scale character of the problem, only methods that take advantage of the special structure of the problem become viable. The Linear Quadratic Regulator algorithm is such an approach in case of linear dynamics. We review it next and we remark that it corresponds to taking a Newton step in view of the previous discussion.

3.2 Linear Quadratic Tracking Problem

We next consider the case of linear dynamics in the optimal control problem of (9) and (10). In the following, we distinguish all variables corresponding to the linear optimal control problem that may have different values in the nonlinear optimal control problem by using an over-bar. When the cost is quadratic as in (11) we have the well-known Linear Quadratic Tracking problem. The control theoretic approach to this problem is based on solving the first order necessary optimality conditions (also sufficient in this case) in an efficient manner by introducing the Riccati equation. We briefly elaborate on this derivation next, for completeness and also since most references assume that the offset sequences $x^o(n)$ and $u^o(n)$ are zero. First, we summarize the first order necessary optimality conditions for this problem.

$$\bar{x}(n+1) = A(n)\bar{x}(n) + B(n)\bar{u}(n) \tag{22}$$

$$\bar{\lambda}^T(n) = [\bar{x}(n) - \bar{x}^o(n)]^T Q(n) + \bar{\lambda}^T(n+1)A(n) \tag{23}$$

$$\partial \bar{J}(n)/\partial \bar{u}(n) = [\bar{u}(n) - \bar{u}^o(n)]^T R(n) + \bar{\lambda}^T(n+1)B(n) \;=\; 0 \tag{24}$$

Note that the system dynamical equations (22) run forward in time $n = 0, \ldots, N-1$ with initial conditions $\bar{x}(0) = \bar{x}_0$ given, while the adjoint dynamical equations (24) run backward in time, $n = N-1, \ldots, 0$ with final conditions $\bar{\lambda}^T(N) = [\bar{x}(N) - \bar{x}^o(N)]^T Q(N)$. From (25), we obtain

$$\bar{u}(n) = \bar{u}^o(n) - R(n)^{-1}B(n)^T\bar{\lambda}(n+1) \tag{25}$$

and by substituting in (22) and (24), we obtain the classical two-point boundary system but with additional forcing terms due to the $\bar{x}^o(n)$ and $\bar{u}^o(n)$ sequences.

$$\begin{aligned}\bar{x}(n+1) &= A(n)\bar{x}(n) - B(n)R(n)^{-1}B(n)^T\bar{\lambda}(n+1) \\ &\quad +B(n)\bar{u}^o(n) \end{aligned} \tag{26}$$

$$\bar{\lambda}^T(n) = Q(n)\bar{x}(n) + A^T(n)\bar{\lambda}(n+1) - Q(n)\bar{x}^o(n) \tag{27}$$

The system of (27) and (27) can be solved by the *sweep method* [3], based on the postulated relation

$$\bar{\lambda}(n) = P(n)\bar{x}(n) + s(n) \tag{28}$$

where $P(n)$ and $s(n)$ are appropriate matrices that can be found as follows. For $n = N$, (28) holds with

$$P(N) = Q(N), \qquad s(N) = -Q(N)\bar{x}^o(N). \tag{29}$$

We now substitute (28) in (27) and after some algebra we obtain

$$\bar{x}(n+1) = M(n)A(n)\bar{x}(n) + v(n) \tag{30}$$

where we defined

$$M(n) = \left[I + B(n)R(n)^{-1}B(n)^T P(n+1)\right]^{-1} \tag{31}$$

$$v(n) = M(n)B(n)[\bar{u}^o(n) - R(n)^{-1}B(n)^T s(n+1)] \tag{32}$$

By replacing $\bar{\lambda}(n)$ and $\bar{\lambda}(n+1)$ in (27) in terms of $\bar{x}(n)$ and $\bar{x}(n+1)$ from (28), we obtain

$$\begin{aligned}P(n)\bar{x}(n) + s(n) = Q(n)\bar{x}(n) + A^T(n)\,[P(n+1)\bar{x}(n+1) \\ +s(n+1)] - Q(n)\bar{x}^o(n),\end{aligned}$$

and by expressing $\bar{x}(n+1)$ from (30) and (32) above, we get

$$\begin{aligned}P(n)\bar{x}(n) + s(n) = \\ Q(n)\bar{x}(n) + A^T(n)P(n+1)M(n)A(n)\bar{x}(n) \\ -A^T(n)P(n+1)M(n)B(n)R(n)^{-1}B(n)^T s(n+1) \\ +A^T(n)P(n+1)M(n)B(n)\bar{u}^o(n) \\ +A^T(n)s(n+1) - Q(n)\bar{x}^o(n)\end{aligned}$$

The above equation is satisfied by taking

$$P(n) = Q(n) + A^T(n)P(n+1)M(n)A(n) \tag{33}$$

$$\begin{aligned}s(n) &= A^T(n)[I - P(n+1)M(n)B(n)R(n)^{-1}B(n)^T]s(n+1) \\ &\quad +A^T(n)P(n+1)M(n)B(n)\bar{u}^o(n) - Q(n)\bar{x}^o(n)\end{aligned} \tag{34}$$

Equation (33) is the well-known Riccati difference equation and together with the auxiliary equation (34), which is unnecessary if $\bar{x}^o(n)$ and $\bar{u}^o(n)$ are zero, are solved backward in time ($n = N-1, \ldots, 1$), with final values given by (29) and together with (31) and (32). The resulting values $P(n)$ and $s(n)$ are stored and used to solve forward in time ($n = 0, \ldots, N-1$), (30) and (25) for the optimal control and state trajectories. These equations are summarized in Table 1.

1. **Solve backward ($n = N-1, \ldots, 0$) with $P_N \equiv Q_N$ and $s_N \equiv -Q_N \bar{x}^o_N$:**

$$\begin{aligned} M(n) &= \left[I + B(n)R(n)^{-1}B(n)^T P(n+1)\right]^{-1} \\ P(n) &= Q(n) + A(n)^T P(n+1) M(n) A(n) \\ s(n) &= A(n)^T \left[I - P(n+1)M(n)B(n)R(n)^{-1}B(n)^T\right] s(n+1) \\ &\quad + A(n)^T P(n+1) M(n) B(n) \bar{u}^o(n) - Q(n)\bar{x}^o(n) \end{aligned}$$

2. **Solve forward ($n = 0, \ldots, N-1$) with $\bar{x}(0) = \bar{x}_0$:**

$$\begin{aligned} v(n) &= M(n)B(n)[\bar{u}^o(n) - R(n)^{-1}B(n)^T s(n+1)] \\ \bar{x}(n+1) &= M(n)A(n)\bar{x}(n) + v(n) \\ \bar{\lambda}(n+1) &= P(n+1)\bar{x}(n+1) + s(n+1) \\ \bar{u}(n) &= \bar{u}^o(n) - R(n)^{-1}B(n)^T \bar{\lambda}(n+1) \end{aligned}$$

Table 1. Algorithm to solve the Discrete-Time Finite-Horizon Linear Quadratic Tracking optimal control problem

3.3 Formulation of the SLQ Algorithm

In the proposed SLQ algorithm, the control at stage $k+1$ is found by performing a one-dimensional search from the control at stage k and along a search direction that is found by solving an Linear Quadratic (LQ) optimal control problem. Specifically, let $U_k = [u(0)\ u(1) \ldots u(N-1)]$ be the optimal solution candidate at step k, and $X_k = [x(1)\ x(2) \ldots x(N)]$ the corresponding state trajectory obtained by solving the dynamical equations (10) using U_k and with the initial conditions $x(0)$. We next linearize the state equations (10) about the nominal trajectory of U_k and X_k. The linearized equations are

$$\bar{x}(n+1) = f_x(x(n),\ u(n))\bar{x}(n) + f_u(x(n),\ u(n))\bar{u}(n) \tag{35}$$

with initial conditions $\bar{x}(0) = 0$. We then minimize the cost index (9) with respect to $\bar{u}(n)$. The solution of this LQ problem gives $\bar{U}_k = [\bar{u}(0)\ \bar{u}(1) \ldots \bar{u}(N-$

1)], the proposed search direction. Thus, the control variables at stage $k+1$ of the algorithm are obtained from

$$U_{k+1} = U_k + \alpha_k \cdot \bar{U}_k \tag{36}$$

where $\alpha_k \in \mathbb{R}^+$ is appropriate stepsize the selection of which is discussed later in the paper. Note again our perspective of considering the optimal control problem as an unconstrained finite-dimensional optimization problem in U.

We emphasize that $\bar{U}_k$ as computed above is *not* the steepest descent direction. It is the solution to a linear quadratic tracking problem for a nonlinear system that has been linearized about U_k. Note that the objective function is not linearized for this solution. Our algorithm is different than standard *Quasilinearization* [3] and *Neighboring Extremal*[18] methods where the adjoint equations are also linearized and two-point boundary problems are solved.

3.4 Properties of the SQL Algorithm

In this section, we prove two important properties of the proposed algorithm. First, we show that search direction $\bar{U}$ is a descent direction.

Theorem 1 *Consider the discrete-time nonlinear optimal control problem of (9) and (10), and assume a quadratic cost function as in (11) and (12) with* $Q(n) = Q^T(n) \geq 0$, $Q(N) = Q^T(N) \geq 0$ *and* $R(n) = R^T(n) > 0$, $n = 0, 1, \ldots, N-1$. *Also consider a control sequence* $U \equiv [u(0)^T \ldots u^T(N-1)]^T$ *and the corresponding state trajectory* $X \equiv [x(1)^T \ldots x^T(N)]^T$. *Next, linearize system (10) about* U *and* X *and solve the following linear quadratic problem:*

$$\begin{aligned} \underset{\bar{u}(n),\bar{x}(n)}{\textit{Minimize}}\ \bar{J} = {} & \frac{1}{2}[\bar{x}^T(N) - \bar{x}^o(N)]Q(N)[\bar{x}(N) + \bar{x}^o(N)] + \\ & \frac{1}{2}\sum_{n=0}^{N-1} \big\{[\bar{x}(n) - \bar{x}^o(n)]^T Q(n)[\bar{x}(n) - \bar{x}^o(n)] \\ & + [\bar{u}(n) - \bar{u}^o(n)]^T R(n)[\bar{u}(n) - \bar{u}^o(n)]\big\} \qquad (37) \end{aligned}$$

subj. to

$$\begin{aligned} \bar{x}(n+1) = {} & f_x(x(n), u(n))\bar{x}(n) + \qquad (38) \\ & f_u(x(n), u(n))\bar{u}(n); \quad \bar{x}(0) = 0, \end{aligned}$$

where $\bar{x}^o(n) \equiv x^o(n) - x(n)$, $\bar{u}^o(n) \equiv u^o(n) - u(n)$. *Then if* $\bar{U} \equiv [\bar{u}(0)^T \ldots \bar{u}^T(N-1)]^T$ *is not zero, it is a descent direction for the cost function (9), i.e.* $J(U + \alpha \cdot \bar{U}) < J(U)$ *for some* $\alpha > 0$.

Proof: We establish that $\bar{U}$ is a descent direction by showing that:

$$\nabla_u J \cdot \bar{U} = \sum_{n=0}^{N-1} \frac{\partial J(n)}{\partial u(n)} \bar{u}(n) < 0, \tag{39}$$

since $\nabla_u J$ in (19) is the gradient of the cost function with respect to the control variables. Now, the components of $\nabla_u J$ are expressed in (19) in terms of the adjoint variables $\lambda(n)$ that satisfy recursion (16) with final values given by (17). On the other hand, $\bar{x}(n)$ and $\bar{u}(n)$ together with adjoint variables $\bar{\lambda}(n)$ satisfy the first order optimality conditions for the linear quadratic problem given in (22), (24) and (25), where $A(n) = f_x(x(n), u(n))$ and $B(n) = f_u(x(n), u(n))$. Let us define

$$\tilde{\lambda}(n) = \bar{\lambda}(n) - \lambda(n) \tag{40}$$

and note from (16) and (24) that

$$\tilde{\lambda}(n)^T = \bar{x}(n)^T Q(n) + \tilde{\lambda}(n+1)^T A(n); \quad \tilde{\lambda}(N) = Q(N)\bar{x}(N). \tag{41}$$

Next through the indicated algebra, we can establish the following relation:

$$\begin{aligned}
&\frac{\partial J(n)}{\partial u(n)} \cdot \bar{u}(n) = \\
&\quad = \Big([u(n) - u^o(n)]^T R(n) + \lambda(n+1)^T B(n) \Big) \bar{u}(n) \\
&\quad\quad \text{(using (19))} \\
&\quad = -\tilde{\lambda}(n+1)^T B(n)\bar{u}(n) - \bar{u}(n)^T R(n)\bar{u}(n) \\
&\quad\quad \text{(using (25))} \\
&\quad = -\tilde{\lambda}(n+1)^T \bar{x}(n+1) + \tilde{\lambda}(n+1)^T A(n)\bar{x}(n) - \bar{u}(n)^T R(n)\bar{u}(n) \\
&\quad\quad \text{(using (39))} \\
&\quad = -\tilde{\lambda}(n+1)^T \bar{x}(n+1) + \tilde{\lambda}(n)^T \bar{x}(n) - \bar{x}(n)^T Q(n)\bar{x}(n) - \\
&\quad\quad \bar{u}(n)^T R(n)\bar{u}(n). \qquad \text{(using (41))}
\end{aligned}$$

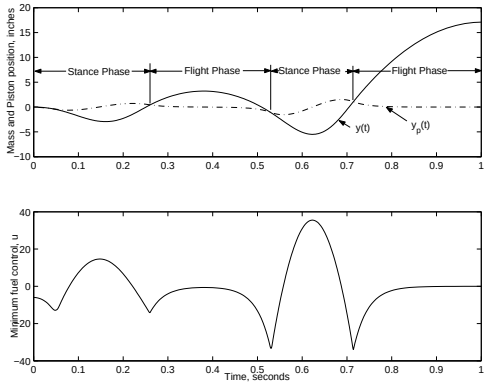
Finally, summing up the above equation from $n = 0$ to $n = N - 1$ and noting that $\bar{x}(0) = 0$ and from (29) that $\bar{\lambda}(N) = Q(N)\bar{x}(N)$, gives:

$$\begin{aligned}
\nabla_u J \cdot \bar{U} &= \sum_{n=0}^{N-1} \frac{\partial J(n)}{\partial u(n)} \cdot \bar{u}(n) \\
&= -\sum_{n=0}^{N-1} [\bar{x}(n)^T Q(n)\bar{x}(n) + \bar{u}(n)^T R(n)\bar{u}(n)] - \\
&\quad \bar{x}^T(N)Q(N)\bar{x}(N) \ < \ 0
\end{aligned} \tag{42}$$

and the proof of the theorem is complete. ■

We remark that the search direction $\bar{U}$ can be found by the LQR algorithm of Table 1 with $A(n) = f_x(x(n), u(n))$ and $B(n) = f_u(x(n), u(n))$.

The next result shows that the proposed SLQ algorithm does in fact converge to a control locally minimizing the cost function (9).



captionMaximum height hopping motion and minimum fuel control.

Theorem 2 *Starting with an arbitrary control sequence U_0, compute recursively new controls:*

$$U_{k+1} = U_k + \alpha_k \cdot \bar{U}_k \tag{43}$$

where the direction U_k is obtained as in Theorem 1 by solving the LQR problem of (37) and the linearized system (39) about the solution U_k and corresponding state trajectory X_k; also α_k is obtained by minimizing $J[U_k+\alpha\bar{U}_k]$ over $\alpha > 0$. Then U_k converges (in the Euclidean norm sense) to a control that locally minimizes the cost function (9) subject to the system equations (10).

Proof: See [14], or note that given the result of the previous theorem, standard convergence proofs (see [19]) apply with either an exact or an inexact linesearch such as the Armijo, Goldstein, or Wolfe search rules [9].

4 Numerical Example

We conducted numerical experiments the hopping system discussed in Case 4, above. We minimized (8) with the following parameters: $k/m = 100$, $g = 386.4$, $\alpha = 0.1$. We assumed that all states were initially zero, and that the initial control sequence was zero. The cost function parameters were selected as: $t_f = 1$, $q = 1000$, $r = 1.0$. As in the last example, a simple Euler approximation was used to discretize the equations, with $N = 50$.

As shown in Figure 4, the algorithm produced an alternating sequence of stance phases and flight phases for the hopping system and it naturally identified the times to switch between these phases. If one were to use collocation methods to solve this problem with explicit consideration of the different dynamics in the different phases, one would have to guess the number of switches between phases and would need to treat the times at which the switch occurs

as variables in the optimization. We note that our algorithm converged much faster when the weighting on the control r is increased; also the number of iterations required for convergence in this problem increases for larger y_N^o, ranging from 3 for $y_N^o = 1$, to 166 for $y_N^o = 50$. In addition, the algorithm failed to converge for $\alpha < 1 \times 10^{-5}$, which demonstrates the need for the dynamics to be continuously differentiable.

5 Conclusion

We discussed the formulation and solution of several important optimal control problems for robotic systems. The most challenging case by far is an underactuated system with contact constraints. We developed an algorithm for solving such nonlinear optimal control problems for systems with quadratic performance measures and unconstrained controls. Contact constraints were accounted for with penalty functions. Each subproblem in the course of the algorithm is a linear quadratic optimal control problem that can be efficiently solved by Riccati difference equations. We show that each search direction generated in the linear quadratic subproblem is a descent direction, and that the algorithm is convergent. Computational experience has demonstrated that the algorithm converges quickly to the optimal solution.

References

1. J.T. Betts, "Survey of Numerical Methods for Trajectory Optimization," *Journal of Guidance, Control and Dynamics,* V. 21: (2) 193-207, 1999.
2. R. W. Brockett, "Robotic manipulators and the product of exponentials formula," *Proc. Int. Symp. Math. Theory of Networks and Systems*, Beer Sheba, Israel, 1983.
3. A.E. Bryson and Y.C. Ho, *Applied Optimal Control,* Wiley, New York, 1995.
4. Y-C. Chen,"Solving robot trajectory Planning problems with uniform cubic B-splines," *Optimal Control Applications and Methods*, Vol. 12, No. 4, pp. 247-262, 1991.
5. A. De Luca, L. Lanari and G. Oriolo,"A Sensitivity Approach to Optimal Spline Robot Trajectories," *Automatica*, Vol. 27, No. 3, pp. 535-539, 1991.
6. von Stryk O. Numerical solution of optimal control problems by direct collocation. *Optimal Control*, Bulirsch R, Miele A, Stoer J, Well KH (eds), *International Series of Numerical Mathematics,* vol. 111. Birkshauser Verlag; Basel, 1993; 129143.
7. Hargraves CR, Paris SW. Direct trajectory optimization using nonlinear programming and collocation. *Journal of Guidance, Control, and Dynamics,* 1987; 10:338342.
8. Enright PJ, Conway BA. Optimal finite-thrust spacecraft trajectories using collocation and nonlinear programming. *Journal of Guidance, Control, and Dynamics,* 1991; 14:981 985.
9. D.G. Luenberger, *Linear and Nonlinear Programming,* Addison Wesley, 1989.

10. B. Martin and J.E. Bobrow, "Minimum Effort Motions for Open Chained Manipulators with Task-Dependent End-Effector Constraints," *International Journal of Robotics Research,* Vol. 18, No. 2, February, 1999, pp. 213-224.
11. R. W. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation*, CRC Press, Boca Raton, Florida, 1993.
12. J.P. Ostrowski, J.P. Desai, V Kumar, "Optimal gait selection for nonholonomic locomotion systems," *International Journal of Robotics Research*, vol.19, (no.3), March 2000. p.225-237.
13. F.C. Park, J.E. Bobrow, and S.R. Ploen, "A Lie Group Formulation of Robot Dynamics," *International Journal of Robotics Research,* Vol. 14, No. 6, December 1995, pp. 609-618.
14. A. Sideris and J. E. Bobrow, "An Efficient Sequential Linear Quadratic Algorithm for Solving Nonlinear Optimal Control Problems," *Proceeding of the 2005 IEEE Conference on Decision and Control,* Portland, Oregon.
15. G. A. Sohl, and J. E. Bobrow, "A Recusive Dynamics and Sensitivity Algorithm for Branched Chains," *ASME Journal of Dynamic Systems, Measurement and Control,* Vol. 123, no. 3, Sept. 2001.
16. G. A. Sohl, and J. E. Bobrow, "Optimal Motions for Underactuated Manipulators," *1999 ASME Design Technical Conferences,* Las Vegas, September.
17. M. Spong, "The swing up control problem for the Acrobot," *IEEE CONTR SYST MAG,* V. 15: (1) 49-55 FEB 1995.
18. Veeraklaew T., and Agrawal S.K., "Neighboring optimal feedback law for higher-order dynamic systems," *ASME J. Dynamic Systems, Measurement, and Control,* 124 (3): 492-497 SEP 2002.
19. R. Fletcher, *Practical Methods of Optimization*, Wiley, 2nd edition, 1987.
20. Chia-Yu E. Wang, Wojciech K. Timoszyk and James E. Bobrow, "Payload Maximization for Open Chained Manipulators: Finding Weightlifting Motions for a Puma 762 Robot," *IEEE Transactions on Robotics and Automation,* April 2001, Vol. 17, No. 2 pp:218-224.
21. C-Y. E. Wang, J. E. Bobrow, and D. J. Reinkensmeyer, "Swinging ¿From The Hip: Use of Dynamic Motion Optimization in the Design of Robotic Gait Rehabilitation," *Proceedings of the IEEE Conference on Robotics and Automation,* Seoul, Korea, May 2001.
22. I. Wickelgren, "Teaching the spinal cord to walk," *Science*, Vol. 279 pp. 319-321, 1998.